



# ORM SQLAlchemy

Object Relational Mapper



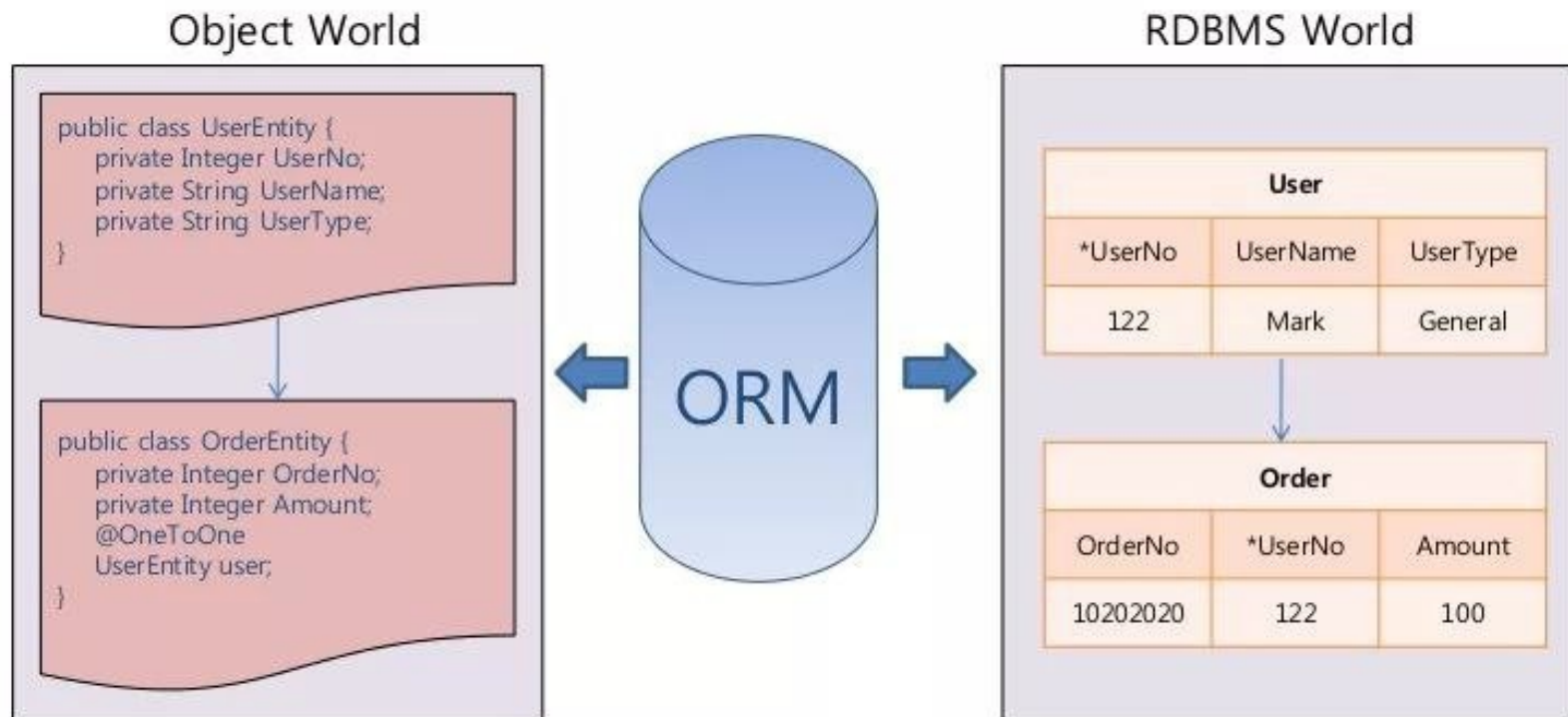
# Определение

Объектно-реляционное отображение (Object-Relational Mapping) — это метод, который позволяет вам запрашивать и манипулировать данными из базы данных, используя объектно-ориентированную парадигму.



# Object Relational Mapping

-Bridge between relational database and object world





## Достоинства ORM

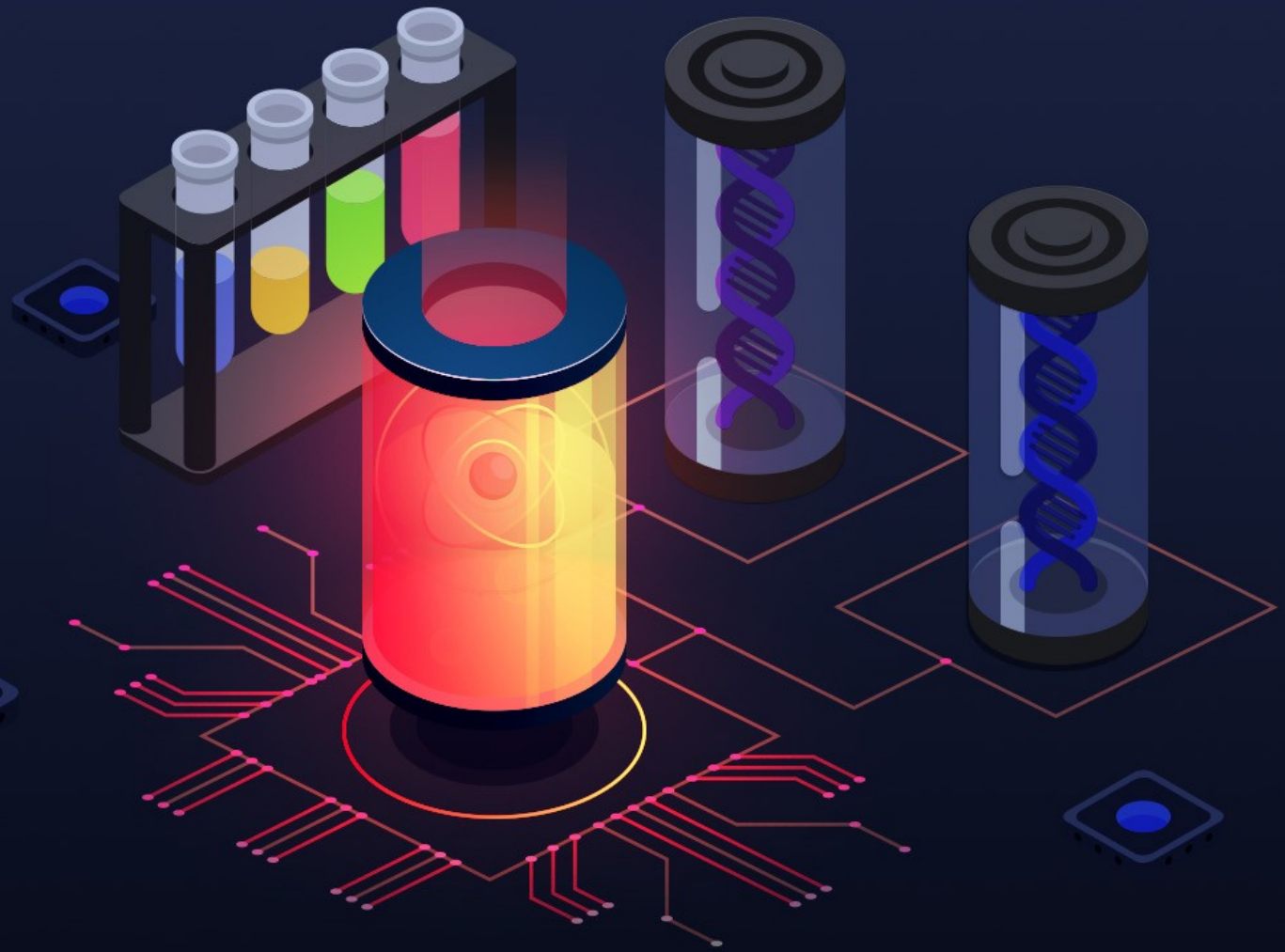
- Безопасность запросов
- Представление параметров типами данных основного языка (без преобразования в типы БД)
- Автоматизация бэкапа и дуплоя
- Прозрачное кеширование данных и возможность выполнения отложенных запросов
- Переносимость (использование разных СУБД (без дополнительных правок в коде)
- Избавление программиста от необходимости вникать в детали реализации той или иной СУБД и синтаксиса соответствующего диалекта языка запросов.



## Недостатки ORM

- Абстрагирование от языка SQL. Во многих случаях выборка данных перестает быть интуитивно понятной и очевидной.
- Дополнительные накладные расходы на конвертацию запросов и создание внутренних объектов самого ORM.
- Возможна потеря производительности

CONSTRUCTING QUERIES WITH  
**SQLALCHEMY**



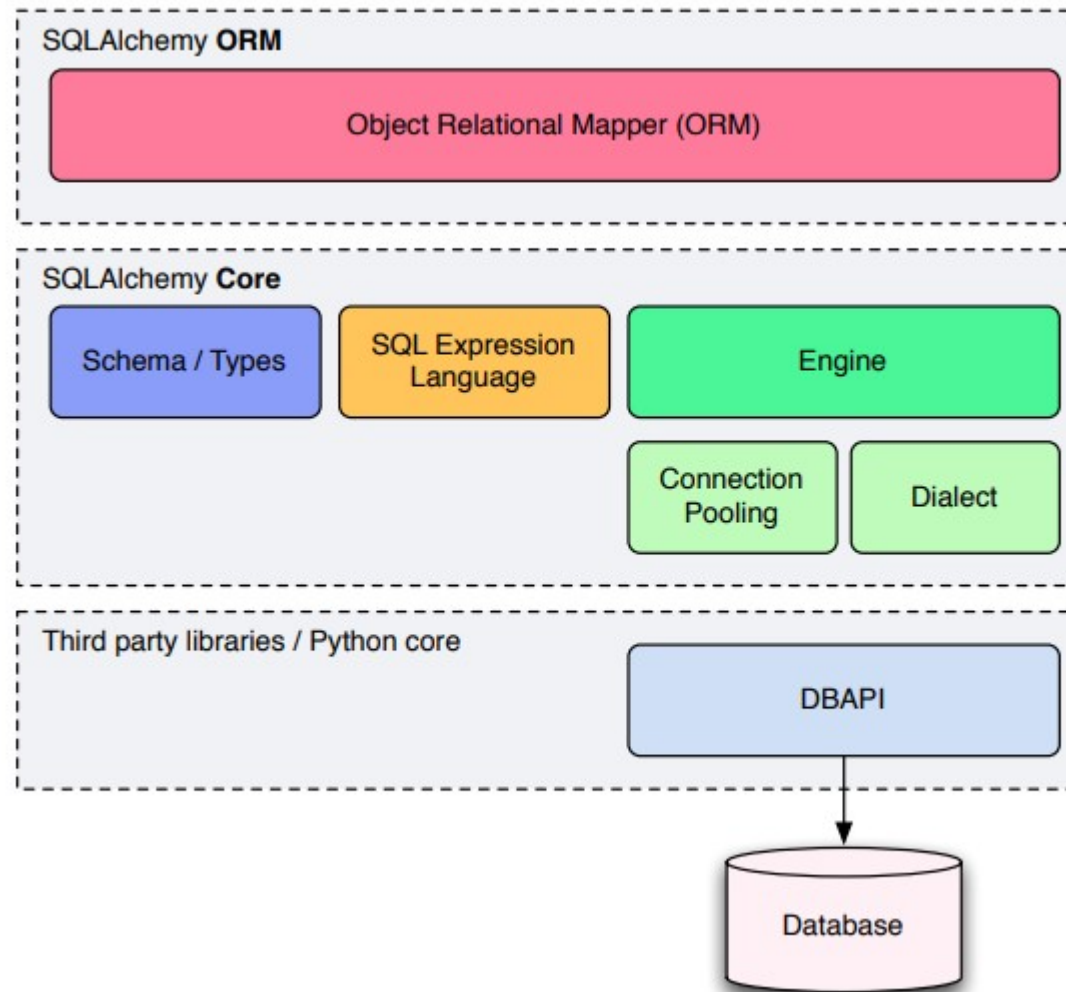


# Философия SQLAlchemy

- Привести использование различных баз данных и адаптеров к максимально согласованному интерфейсу
- Никогда не "скрывать" базу данных или ее концепции разработчики должны знать / продолжать думать на языке SQL.
- Обеспечить автоматизацию рутинных операций CRUD
- Разрешить выразить синтаксис DB/SQL в декларативном шаблоне.



# Архитектура SQLAlchemy



SQLAlchemy состоит из двух компонентов ORM и CORE



## Компонента SQLAlchemy - Core

SQLAlchemy Core - это абстракция над традиционным SQL. Он предоставляет SQL Expression Language, позволяющий генерировать SQL-инструкции с помощью конструкций Python.

- **Engine** - механизм, который обеспечивает подключение к конкретному серверу базы данных.
- **Dialect** - интерпретирует разные диалекты SQL и команды базы данных в синтаксис конкретного DBAPI и серверной части базы данных.
- **Connection Pool** - хранит коллекцию подключений к БД для быстрого повторного использования
- **SQLExpression Language** - позволяет писать SQL запрос с помощью выражений Python
- **Schema/Types** - использует объекты Python для представления таблиц, столбцов и типов данных.



# SQLAlchemy - ORM

- Позволяет создавать объекты Python, которые могут быть сопоставлены с таблицами реляционной базы данных
- Предоставляет систему запросов, которая загружает объекты и атрибуты с использованием SQL, сгенерированного на основе сопоставлений.
- Выстроена поверх Core - использует Core для создания SQL и обращений с базой данных
- Представляет несколько более объектно-ориентированную перспективу, в отличие от перспективы, ориентированной на схему



# The Python DBAPI

- DBAPI PEP-0249, Python Database API
- Система де-факто для предоставления интерфейсов баз данных Python
- Существует множество доступных реализаций DBAPI, большинство баз данных имеют более чем одну




# The Python DBAPI

```
import psycopg2
connection = psycopg2.connect("scott", "tiger", "test")

cursor = connection.cursor()
cursor.execute(
    "select emp_id, emp_name from employee "
    "where emp_id=%(emp_id)s",
    {'emp_id':5})
emp_name = cursor.fetchone()[1]
cursor.close()

cursor = connection.cursor()
cursor.execute(
    "insert into employee_of_month "
    "(emp_name) values (%(emp_name)s)",
    {"emp_name":emp_name})
cursor.close()

connection.commit()
```



# Как мы можем гипотетически представить декларативный стиль описание структур БД

```
# a hypothetical declarative system

class User(ORMObject):
    tablename = 'user'

    name = String(length=50)
    fullname = String(length=100)

class Address(ORMObject):
    tablename = 'address'

    email_address = String(length=100)
    user = many_to_one("User")
```



Для работы с ORM SQLAlchemy нужно  
установить библиотеку для Flask

```
>pip install -r Flask-SQLAlchemy
```

<https://flask-sqlalchemy.palletsprojects.com/>

# Создание таблицы БД через объект

```
# файл model.py
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://postgres:postgres@localhost:5432'

db = SQLAlchemy(app)
class User(db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True,
nullable=False)
    email = db.Column(db.String(120), unique=True,
nullable=False)

    def __repr__(self):
        return '<User %r>' % self.username
db.create_all() - ?
```





## Создание объекта и запись в БД

```
# импортируем из файла model.py
from model import User, db
admin = User(username='admin',
mail='admin@example.com')
guest = User(username='guest', email='guest@example.com')
db.session.add(admin)
db.session.add(guest)
db.session.commit()
```



# Получение пользователя из БД

```
User.query.get(1)
```

```
<User u'admin'>
```

```
User.query.all()
```

```
[<User u'admin'>, <User u'guest'>]
```

```
User.query.filter_by(username='admin').first()
```

```
<User u'admin'>
```

Подробнее по методам которые соответствуют операциям  
SELECT, INSERT, UPDATE, DELETE см. документацию

<https://docs.sqlalchemy.org/en/14/core/tutorial.html>

<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>



# Типы данных

- `Integer()` - basic integer type, generates `INT`
- `String()` - ASCII strings, generates `VARCHAR`
- `Unicode()` - Unicode strings - generates `VARCHAR`, `NVARCHAR` depending on database
- `Boolean()` - generates `BOOLEAN`, `INT`, `TINYINT`
- `DateTime()` - generates `DATETIME` or `TIMESTAMP`, returns Python `datetime()` objects
- `Float()` - floating point values
- `Numeric()` - precision numerics using Python `Decimal()`



# Отношение один ко многим

```
# Child
class Member(db.Model):
    __tablename__ = 'member'

    id = db.Column(db.Integer, primary_key=True)
    id_role = db.Column(db.ForeignKey('role.id'))

# Parent
class Role(db.Model):
    __tablename__ = 'role'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String)
    describe = db.Column(db.Text)
    members = db.relationship('Member', backref='role')
```



## Прямые запросы к БД.

```
connection = db.session.connection()  
connection.execute( <sql here> )
```

```
from sqlalchemy import text  
sql = text('select name from penguins')  
result = db.engine.execute(sql) - ?  
names = [row[0] for row in result]  
print names
```



## Key ORM Patterns

- **Unit of Work** - Сессии в рамках этого паттерна отслеживают изменения, сделанные в рамках одной бизнес-транзакции, а затем “сбрасывают” их пачкой в базу, предварительно выполнив топологическую сортировку по зависимостям и сгруппировав повторяющиеся операции.
- **Identity Map** - объекты отслеживаются по их первичному ключу .
- **Lazy Loading** - Некоторые атрибуты объекта могут выдавать дополнительные SQL-запросы при обращении к ним.
- **Eager Loading** - Для загрузки связанных объектов и коллекций одновременно запрашивается несколько таблиц.
- **Method Chaining** - Запросы составляются с использованием строки вызовов методов, каждый из которых возвращает новый объект запрос

